Python 数据分析与数据挖掘（Python for Data Analysis&Data Mining）

Chap 8 回归： 数值型数据预测（Linear Regression）

内容：

- 回归概述
- 最佳拟合直线
- 局部加权线性回归
- 缩减系数来“理解”数据
- 影响因素分析和预测建模
- SKlearn实现Linear Regression
- 算法：最小二乘法最佳拟合直线，局部加权线性回归（lwlr），岭回归，lasso，前向逐步回归等算法性能对比
- 应用领域：预测销售量、预测制造缺陷率、预测产品价格、预测年龄、预测股价、预测GDP等

实践：

- 实例1：预测鲍鱼的年龄
- 实例2：台风路径预测

作业**3**

- 台风路径预测：运用多种线性回归策略提高台风路径预测的准确性

这节课是在前面数据分析的基础上，对数值型数据进行分析和建模并预测连续型的数值。

准备工作：导入库，配置环境等

In [1]:
```
from __future__ import division
import os, sys

# 导入库并为库起个别名
import numpy as np
import pandas as pd
from pandas import Series, DataFrame

# 启动绘图
%matplotlib inline
import matplotlib.pyplot as plt

# 常用全局配置
np.random.seed(12345)
np.set_printoptions(precision=4)
plt.rc('figure', figsize=(10, 6))
```

```
br = '\n'
```

1. 回归概述

线性回归的主要目的是预测连续型数值的目标变量值，例如GDP、股价、销售量、年龄、房价等数值型数据。

最直接的办法就是依据输入写出一个目标值的计算公式。例如要预测下一部手机的价格，可能会这样计算：

手机价格 = 0.01 年薪 + *0.15* 当前手机价格 - 0.99 * 年龄 + 100

这个线性回归方程（regression equation）中，0.01，0.15，-0.99和100都称为回归系数（regression weights），求解这些回归系数的过程就是回归。

优点：结果易于理解，计算不复杂。

缺点：对非线性的数据拟合不好。

适用数据类型：数值型和标称型数据。是对分类方法的提升，可以预测连续型数据而不仅仅是离散的类别标签。

对于非线性回归模型，可能计算公式写成：

手机价格 = 0.1 * 年薪 / 年龄

回归的一般方法过程

- 收集数据
- 准备数据：回归需要数值型数据，标称型数据被转成二值型数据
- 分析数据：绘出数据的可视化二维图有助于对数据做出理解和分析，求得新回归系数后，可以将新拟合线进行对比
- 训练算法：找到回归系数
- 测试算法： 使用预测值和数据的拟合度，来分析模型的效果
- 使用算法： 使用回归，在给定输入的时候预测出一个数值。

2. 最佳拟合直线

关键问题：如何求解数据集的回归系数**w**?

假定输入数据x存放在矩阵X中，而回归系数存放在向量w中，那么对于给定的数据x1， 预测结果可以通过 $y_{1} = x_{1}^T * w$ 得出。给定一些x和对应的y，怎样才能找到w呢？

- 一个常用的方法就是找出使误差最小的w。 误差 = 预测y和真实y值之间的差值，使用该误差的简单累加将使得正差值和负差值相互抵消，因此，我们采用平方误差，写作：$\sum_{i=1}^m (y_{i} - x_{i}^T * w)^{2}$， 用矩阵表示可以写作: $(y-Xw)^{T} (y-Xw)$。

如果对w求导，得到：$(X)^{T} (Y-Xw)$， 令其等于零，解出w如下： $ \hat{w}= (X^{T}X)^{-1}X^{T}y$

w上面的$\hat{w}$ 表示这是当前可以估计出的w的最优解。此外，公式中包含 $(X^{T}X)^{-1}$，需要对矩阵求逆，因此，这个方程只在逆矩阵存在的时候使用。然而，矩阵的逆可能并不存在，因此必须要在代码中对此作出判断。

最佳w求解是统计学中的常见问题，除了矩阵方法还有很多其他方法。通过调用Numpy库中的矩阵方法，仅使用几行代码就可以完成所需功能。该方法也称作OLS（ordinary least squares，普通最小二乘法）。

In [2]:
```
from __future__ import division
import os, sys
from numpy import *
from math import *

#加载数据
def loadDataSet(fileName):       #general function to parse tab -delimited floats
    numFeat = len(open(fileName).readline().split('\t')) - 1 #get number of fi
elds
    dataArr = []
    labelArr = []
    fr = open(fileName)
    for line in fr.readlines():
        lineArr =[]
#     lineArr.append(1.0) # 增加这一列是为了进行线性回归，存放常量x0
        curLine = line.strip().split('\t')
        for i in range(numFeat):
            lineArr.append(float(curLine[i]))
        dataArr.append(lineArr)
        labelArr.append(float(curLine[-1]))  # 最后一项是类标
    return dataArr,labelArr
```

In [3]:
```
# 标准回归训练
def standRegres(xArr,yArr):
    xMat = mat(xArr); yMat = mat(yArr).T  # list转换成mat
    xTx = xMat.T * xMat
    if linalg.det(xTx) == 0.0:  # 矩阵的逆可能并不存在，因此必须要在代码中对此作
出判断
        print "This matrix is singular, cannot do inverse"
        return
    ws = xTx.I * (xMat.T * yMat)
    return ws
```

In [4]:
```
# 执行windows命令显示文件，如果是unix，使用cat
#!type data\ex0.txt
```

In [5]:
```
xArr, yArr = loadDataSet("data/ex0.txt")
```

In [6]:
```
# 看前两条数据
xArr[0:2]
# 第一个值总是等于1.0， 即x0，我们假定偏移量是一个常数；第二个值是x1
```

Out[6]:
```
[[1.0, 0.067732], [1.0, 0.42781]]
```

In [7]:
```
# 前两条数据的真实值
yArr[0:2]
```

Out[7]:
```
[3.176513, 3.816464]
```

In [8]:
```
# 执行标准回归训练函数
ws = standRegres(xArr,yArr)
```

In [9]:
```
# 变量ws就是回归系数
```

```
ws
```

Out[9]:
```
matrix([[ 3.0077],
        [ 1.6953]])
```

用内积来预测y，第一维 * 常数x0， 第二维 * 输入变量x1， 前面假定了x0=1， 所以最终得到： $y = ws[0] + ws[1]*x1$

注意：这里的y实际是预测出来的，为了和真实的y区分开，我们记预测y为 : $\hat{y}$， 即 yHat。

下面使用新的ws值计算yHat

In [10]:
```
# 使用新的ws值计算yHat
xMat = mat(xArr)   # list转换成mat
yMat = mat(yArr)   # list转换成mat
yHat = xMat * ws
```

In [11]:
```
print yArr[0:2] # 前两条数据的真实值
print yHat[0:2] # 前两条数据的预测值
```

```
[3.176513, 3.816464]
[[ 3.1226]
 [ 3.733 ]]
```

In [12]:
```
yHat[0:2].tolist()  # mat转换成list
```

Out[12]:
```
[[3.1225708358971276], [3.7330192222448586]]
```

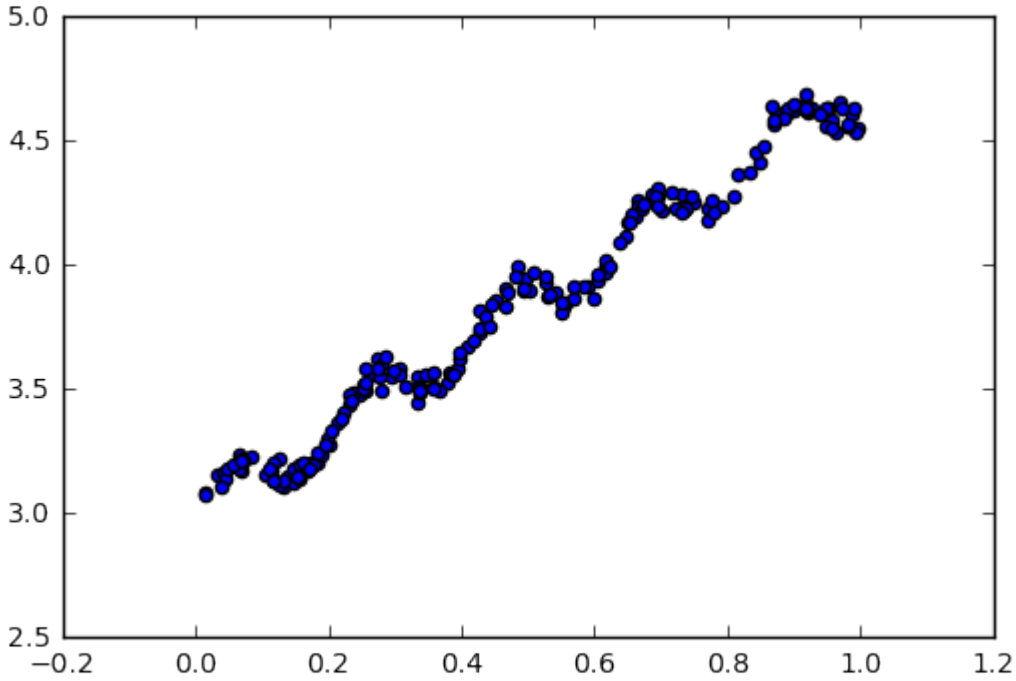
绘制数据集的散点图和最佳拟合直线图

数据可视化便于观察回归直线与原始数据的拟合情况

In [13]:
```
# 绘制数据集的散点图
%matplotlib inline
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(xMat[:, 1].flatten().A[0], yMat.T[:, 0].flatten().A[0]) # 原始数据
```

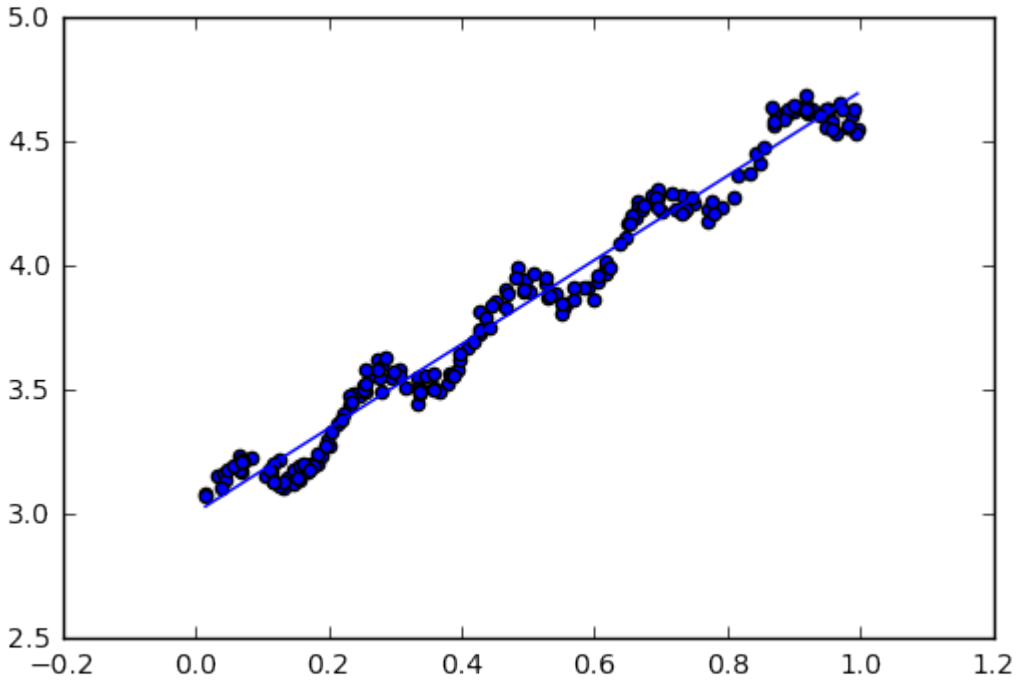
Out[13]:
```
<matplotlib.collections.PathCollection at 0xa227358>
```



In [14]:
```
# 如果直线上的数据点词序混乱，绘图时将会出现问题，所以首先将点按照升序排列
xCopy = xMat.copy()
xCopy.sort(0)
yHat = xCopy * ws

# 绘制最佳拟合直线图，需要绘出yHat的值
fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(xMat[:, 1].flatten().A[0], yMat.T[:, 0].flatten().A[0]) # 原始数据
ax.plot(xCopy[:, 1], yHat) # 拟合直线
```

Out[14]:
```
[<matplotlib.lines.Line2D at 0xa2d8fd0>]
```



怎么计算预测值yHat序列和真实值y序列的匹配程度？

解决方法：计算这两个序列的相关系数

- Python中的NumPy提供了corrcoef函数可以计算预测值和真实值的相关性， corrcoef(yHat, yActual)
- yHat需要转置，以保证两个向量都是行向量

In [15]:
```
# 首先计算出y的预测值yHat
yHat = xMat * ws
```

In [16]:
```
# 然后计算相关系数（需要将yMat转置，以保证两个向量都是行向量）
corrcoef(yMat, yHat.T)   # yMat=mat(yArr) yMat是个行向量
```

Out[16]:
```
array([[ 1.    ,  0.9865],
       [ 0.9865,  1.    ]])
```

结果：得到的矩阵包括所有两两组合的相关系数。yHat和yMat的相关系数为0.98。最佳拟合直线方法将数据视为直线进行建模，具有不错的表现。但是对比原始数据的散点图和最佳拟合直线图，似乎数据中还存在其他的潜在模式。

如何才能利用这些模式呢？

- 可以根据数据来局部调整预测，即下面的采用局部加权线性回归

3. 局部加权线性回归

线性回归的一个问题是可能出现欠拟合，因为它求的是具有最小均方误差的无偏估计。因而，一些方法允许在估计中引入一些偏差，从而降低预测的均方误差。

局部加权线性回归（Locally Weighted Linear Regression， LWLR）是其中之一。给待预测点附近的每个点赋予一定的权重，然后在这个子集上基于最小均方差来进行普通的回归。与kNN一样，这个算法每次预测均需要事先选取出对应的数据子集。

该算法解出的回归系数w的形式如下： $ \hat{w} = (X^{T}WX)^{-1}X^{T}Wy $, 其中W是一个权重矩阵，用来给每个数据点赋予权重。

回顾前面最小二乘法求解w为： $ \hat{w}= (X^{T}X)^{-1}X^{T}y$

LWLR使用核（与SVM中的核类似）来对附近的点赋予更高的权重。核的类型可以自由选择，最常用的核就是高斯核，对应的权重为： $w(i,i)=exp(\frac{| x^{(i)}- x |}{-2k^{2}})$， 其中包含一个需要用户指定的参数k，决定对附近的点赋予多大的权重，这也是LWLR时唯一需要考虑的参数。

In [17]:
```
# 局部加权线性回归训练
def lwlr(testPoint,xArr,yArr,k=1.0):
    xMat = mat(xArr); yMat = mat(yArr).T
    m = shape(xMat)[0]  # 样本个数
    weights = mat(eye((m)))  # 创建对角矩阵weights，是个方阵，阶数等于样本个数，为每个样本点初始化一个权重
    for j in range(m):      # 遍历数据集
        diffMat = testPoint - xMat[j,:]     #计算每个样本点对应的权重值：随着样本点与待预测点距离的递增
        weights[j,j] = exp(diffMat*diffMat.T/(-2.0*k**2))  # 权重值大小以指数级衰减，k控制衰减的速度
    xTx = xMat.T * (weights * xMat)
    if linalg.det(xTx) == 0.0:
        print "This matrix is singular, cannot do inverse"
        return
```

```
    ws = xTx.I * (xMat.T * (weights * yMat)) #得到回归系数ws的一个估计
    return testPoint * ws

def lwlrTest(testArr,xArr,yArr,k=1.0):  #为数据集中每个点调用lwlr（），有助于求解k的大小
    m = shape(testArr)[0]
    yHat = zeros(m)
    for i in range(m):
        yHat[i] = lwlr(testArr[i],xArr,yArr,k)
    return yHat

def lwlrTestPlot(xArr,yArr,k=1.0):  #same thing as lwlrTest except it sorts X first
    yHat = zeros(shape(yArr))       #easier for plotting
    xCopy = mat(xArr)
    xCopy.sort(0)
    for i in range(shape(xArr)[0]):
        yHat[i] = lwlr(xCopy[i],xArr,yArr,k)
    return yHat,xCopy
```

In [18]:
```
#加载数据
xArr, yArr = loadDataSet("data/ex0.txt")
```

In [19]:
```
#对单点进行预测估计
print yArr[0]
print lwlr(xArr[0], xArr, yArr, 1.0)
print lwlr(xArr[0], xArr, yArr, 0.05) #改变k参数的大小
print lwlr(xArr[0], xArr, yArr, 0.001) #改变k参数的大小
```

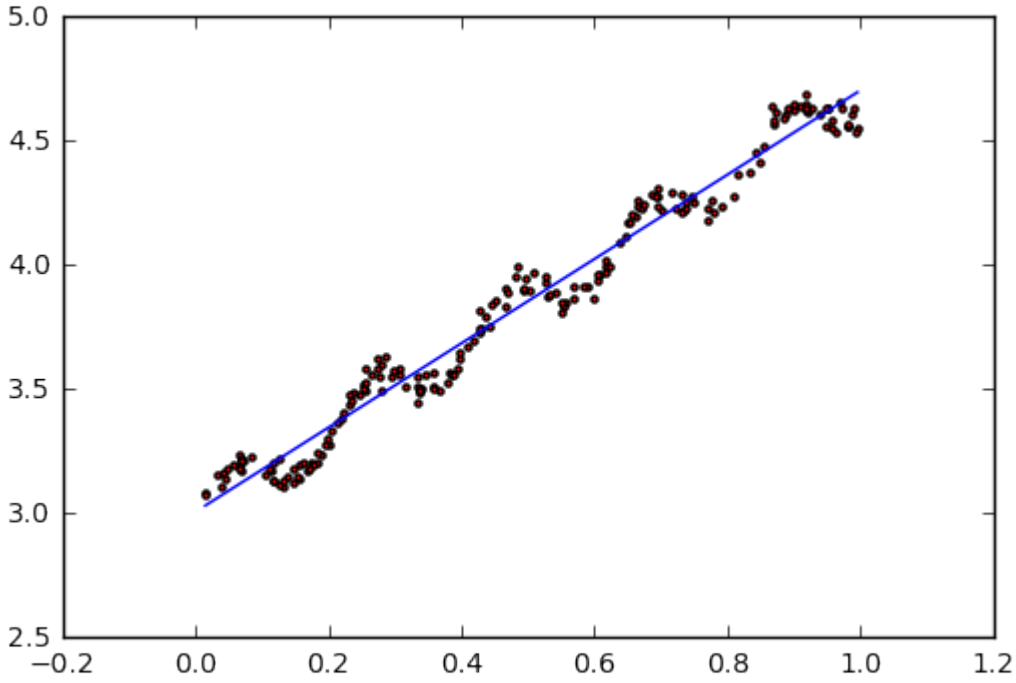
```
3.176513
[[ 3.122]]
[[ 3.1728]]
[[ 3.2018]]
```

In [20]:
```
#为了得到数据集中所有点的估计，调用lwlrTest（）函数
yHat = lwlrTest(xArr, xArr, yArr, 1.0)
```

In [21]:
```
#绘图观察估计值和原始值的拟合效果
#首先将数据点xArr按序排列
xMat = mat(xArr)
srtInd = xMat[:,1].argsort(0)
xSort = xMat[srtInd][:,0,:]

#绘制最佳拟合直线图，需要绘出yHat的值
fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(xMat[:, 1].flatten().A[0], yMat.T[:, 0].flatten().A[0],s=5, c='red') #原始数据
ax.plot(xSort[:,1], yHat[srtInd]) #拟合直线
```

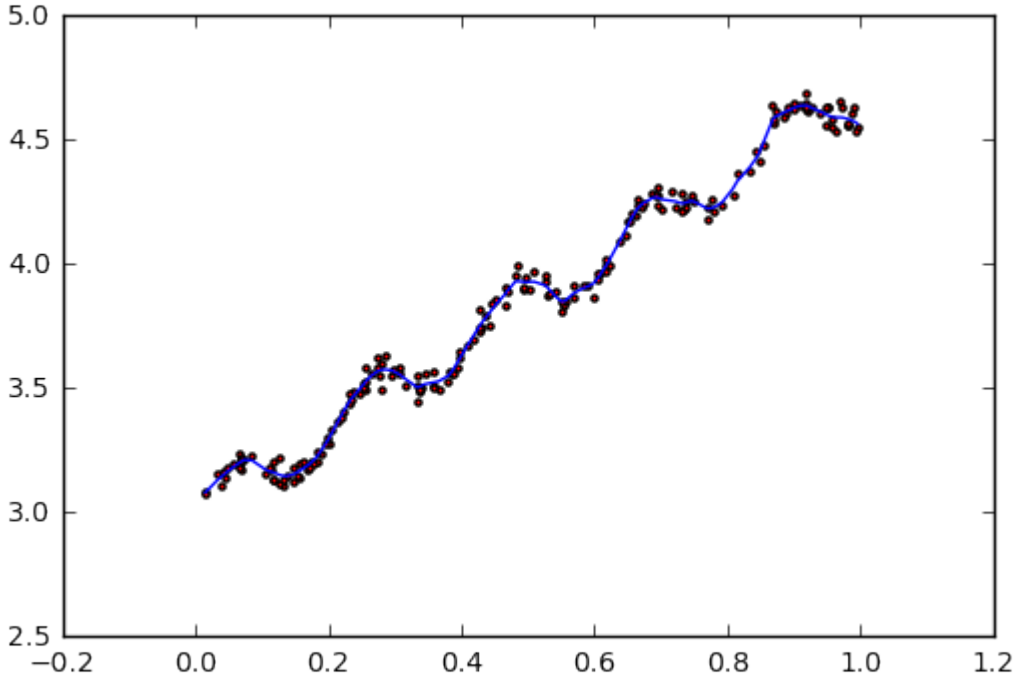
Out[21]:
```
[<matplotlib.lines.Line2D at 0xa609a90>]
```



In [22]:
```
#改变k参数的值， k=0.01
yHat = lwlrTest(xArr, xArr, yArr, 0.01)
#绘图观察估计值和原始值的拟合效果
#首先将数据点xArr按序排列
xMat = mat(xArr)
srtInd = xMat[:,1].argsort(0)
xSort = xMat[srtInd][:,0,:]

#绘制最佳拟合直线图，需要绘出yHat的值
fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(xMat[:, 1].flatten().A[0], yMat.T[:, 0].flatten().A[0], s=5,
c='red') #原始数据
ax.plot(xSort[:,1], yHat[srtInd]) #拟合直线
```

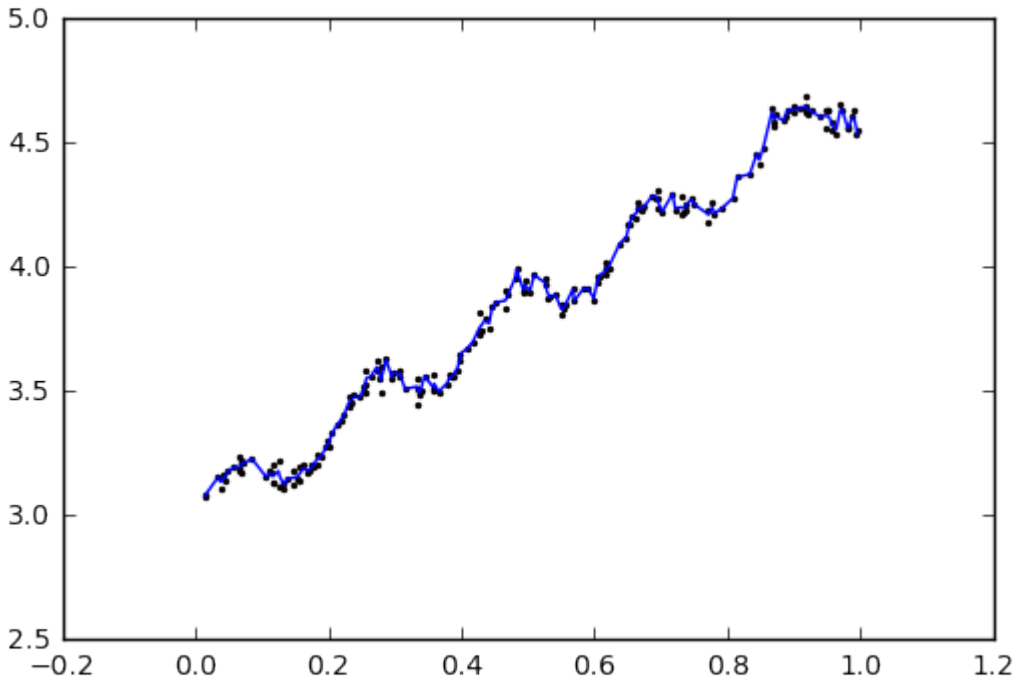
Out[22]:
```
[<matplotlib.lines.Line2D at 0xaa4bf60>]
```



In [23]:
```
#改变k参数的值， k=0.003
yHat = lwlrTest(xArr, xArr, yArr, 0.003)
#绘图观察估计值和原始值的拟合效果
#首先将数据点xArr按序排列
xMat = mat(xArr)
srtInd = xMat[:,1].argsort(0)
xSort = xMat[srtInd][:,0,:]

#绘制最佳拟合直线图，需要绘出yHat的值
fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(xMat[:, 1].flatten().A[0], yMat.T[:, 0].flatten().A[0], s=2,
c='red') #原始数据
ax.plot(xSort[:,1], yHat[srtInd]) #拟合直线
```

Out[23]:
```
[<matplotlib.lines.Line2D at 0xa7c9c50>]
```



观察： 平滑参数k的不同值的拟合效果

- k=1.0时的模型效果与最小二乘法差不多
- k=0.01时该模型效果可以挖出数据的现在规律
- k=0.003时则考虑太多的噪声，进而导致了过拟合现象（overfitting）

局部加权线性回归存在一个问题：计算量增加，因为对每个点做预测时都必须要使用整个数据集。也就是说为了做出预测，必须保存所有的训练数据。

实例**1**： 预测鲍鱼的年龄

鲍鱼是一种介壳类水生动物，它的年龄可以从鲍鱼壳的层数推算得到

In [24]:
```
#计算误差平方和 square sum of error (SSE),
def rssError(yArr,yHatArr): #yArr and yHatArr both need to be arrays
    return ((yArr-yHatArr)**2).sum()
```

In [25]:
```
abX, abY = loadDataSet('data/abalone.txt')
```

In [26]:
```
yHat01 = lwlrTest(abX[0:99], abX[0:99],abY[0:99], 0.1)
yHat1 = lwlrTest(abX[0:99], abX[0:99],abY[0:99], 1)
yHat10 = lwlrTest(abX[0:99], abX[0:99],abY[0:99], 10)
```

In [27]:
```
#分析预测误差的大小，可以用函数rssError计算出这个指标
err01 = rssError(abY[0:99], yHat01.T)
err1 = rssError(abY[0:99], yHat1.T)
err10 = rssError(abY[0:99], yHat10.T)
print err01, err1, err10
```

```
56.8041972556 429.89056187 549.118170883
```

使用较小的核将得到较低的误差，那么为什么不在所有数据集上都使用最小的核呢？因为使用过小的核会造成过拟合，对新数据不一定能达到最好的预测效果。

下面在新数据上检查一下模型的表现：

- 在上面的三个参数中，核为10的测试误差最小，但是在训练集的误差确实最大。

In [28]:
```
yHat01 = lwlrTest(abX[100:199], abX[0:99],abY[0:99], 0.1)
yHat1 = lwlrTest(abX[100:199], abX[0:99],abY[0:99], 1)
yHat10 = lwlrTest(abX[100:199], abX[0:99],abY[0:99], 10)

#分析预测误差的大小，可以用函数rssError计算出这个指标
err01 = rssError(abY[100:199], yHat01.T)
err1 = rssError(abY[100:199], yHat1.T)
err10 = rssError(abY[100:199], yHat10.T)
print err01, err1, err10
```

```
106085.406572 573.52614419 517.571190538
```

在上面的三个参数中，核为10的测试误差最小，但是在训练集的误差却是最大。

In [29]:
```
#接下来再和简单线性回归做个比较:
ws = standRegres(abX[0:99],abY[0:99])
yHat = mat(abX[100:199]) * ws
rssError(abY[100:199], yHat.T.A)
```

Out[29]:
```
518.63631532464444
```

** 观察：

- 简单线性回归达到了与局部加权线性回归类似的效果。这也表明，必须在未知数据上比较效果才能选取到最佳模型。
- 那么，怎么选取最佳的核k值？可以采用交叉验证方法，用10个不同的样本集做10次测试来比较结果。

下面介绍另一种提高预测精度的方法。

4. 缩减系数来“理解”数据

如果数据的特征比样本点个数还多怎么办？ 还使用前面的线性回归和之前的方法做预测，是不可以的！

原因： small big data （特征比样本数多），即列 > 行 （n > m）时，前面的线性回归方法做预测是无效的。因为在计算 $(X^{T}X)^{-1}$ 时候会出错。此时，输入数据的矩阵x不是满秩矩阵，而非

满秩矩阵在求逆时会出现问题。

为了解决这个问题，引入了岭回归（ridge regression）。

第一种缩减方法：岭回归

第二种缩减方法：lasso法 （效果很好但是计算复杂），因此介绍前向逐步回归，得到与lasso相当的效果，且更容易实现。

4.1 岭回归

做法： 在矩阵 $ X^{T}X $ 上加一个 $ \lambda I $ ，从而使得矩阵非奇异，进而能对 $ X^{T}X + \lambda I $ 求逆。 其中 $ I $ 是个 m * m的单位矩阵，$ \lambda $ 是用户定义数值，这样，回归系数w的计算公式变为： $ \hat{w} = (X^{T}X + \lambda I)^{-1}X^{T}y $

岭回归起初用于处理特征数多于样本数的情况，现在也用于在估计中加入偏差，从而得到更好的估计。这里通过引入 $ \lambda $ 来限制了所有w之和，通过引入这个惩罚项，能减少不重要的参数（从而能更好的理解数据），这个技术叫做缩减（shrinkage）。因此，与简单线性回归相比，缩减法能取得更好的预测效果。

对参数lambda的训练，也是通过预测误差最小化得到：选取不同的labmda来进行测试，最终得到一个使预测误差最小的lambda的值。

In [30]:
```
#岭回归训练
def ridgeRegres(xMat,yMat,lam=0.2):  #默认lambda=0.2
    xTx = xMat.T * xMat  #构建矩阵xTx
    denom = xTx + eye(shape(xMat)[1])*lam  #然后加上lambda乘以单位矩阵，调用eye生成
    if linalg.det(denom) == 0.0:  #如果lambda设为0还会产生错误，扔需要检查矩阵非奇异，行列式是否为0
        print "This matrix is singular, cannot do inverse"
        return
    ws = denom.I * (xMat.T*yMat)  #计算回归系数
    return ws

#岭回归在一组lambda上测试结果
def ridgeTest(xArr,yArr):
    xMat = mat(xArr); yMat=mat(yArr).T
    yMean = mean(yMat,0)  #数据标准化，（特征-均值）/方差
    yMat = yMat - yMean     #to eliminate X0 take mean off of Y
    #regularize X's
    xMeans = mean(xMat,0)   # 数据标准化 calc mean then subtract it off
    xVar = var(xMat,0)      #calc variance of Xi then divide by it
    xMat = (xMat - xMeans)/xVar
    numTestPts = 30
    wMat = zeros((numTestPts,shape(xMat)[1]))  #
    for i in range(numTestPts):      #取30个不同的lambda值，调用ridgeRegres
        ws = ridgeRegres(xMat,yMat,exp(i-10))  #lambda取指数级变化，可以看出lambda在取非常小和非常大值时的影响
        wMat[i,:]=ws.T
    return wMat
```

In [31]:
```
#预测鲍鱼年龄
abX,abY = loadDataSet('data/abalone.txt')
```

In [32]:
```
abX[0:2]
```

Out[32]:
```
[[1.0, 0.455, 0.365, 0.095, 0.514, 0.2245, 0.101, 0.15],
 [1.0, 0.35, 0.265, 0.09, 0.2255, 0.0995, 0.0485, 0.07]]
```

In [33]:
```
abY[0:2]
```

Out[33]:
```
[15.0, 7.0]
```

In [34]:
```
ridgeWeights = ridgeTest(abX, abY)
ridgeWeights[0:2]
```

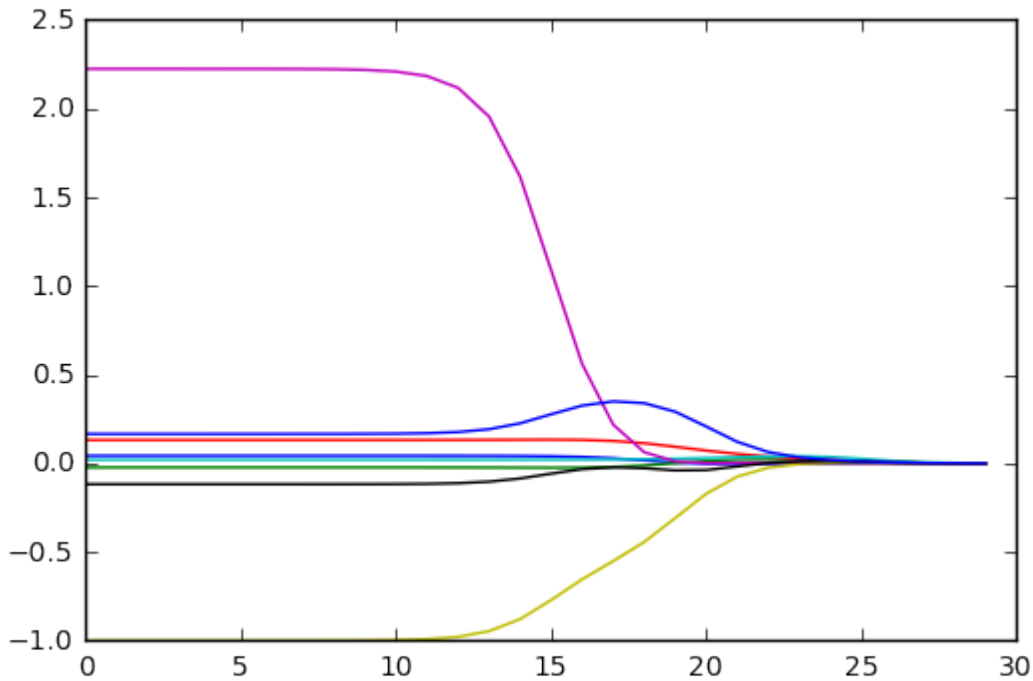
Out[34]:
```
array([[ 0.043 , -0.0227,  0.1321,  0.0208,  2.224 , -0.999 , -0.1173,
         0.1662],
       [ 0.043 , -0.0227,  0.1321,  0.0208,  2.224 , -0.999 , -0.1173,
         0.1662]])
```

在**30**个不同的**lambda**下调用**ridgeRegres**函数，这里的**lambda**以指数级变化，可以看出**lambda**在取非常小的值和非常大的值时分别对结果造成的影响。

为了看到缩减的效果，使用下面的绘图：

In [35]:
```
# 启动绘图
%matplotlib inline
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(ridgeWeights)
plt.show()
```



观察：

x轴是**log(lambda)**的值从**0**到**30**， **y**轴是对应的**8**个回归系数的值的变化。

- 最左边，即lambda最小时，可以得到所有系数的原始值，和普通回归一样；
- 在最右边，lambda很大时，所有回归系数缩减为0；
- lambda取中间某值时，将可以取到最好的预测效果。

how？ CV 为了找到最佳参数值，需要进行交叉验证。

要判断哪些变量对结果预测最具有影响力，观察它们对应的系数大小即可。

In [36]:
```
RGws01 = ridgeRegres(mat(abX[0:99]),mat(abY[0:99]).T,0.1)  # 默认lambda=0.2
RGws02 = ridgeRegres(mat(abX[0:99]),mat(abY[0:99]).T,0.2)  # 默认lambda=0.2
RGws05 = ridgeRegres(mat(abX[0:99]),mat(abY[0:99]).T,0.5)  # 默认lambda=0.2

yHat01 = mat(abX[100:199]) * RGws01
yHat02 = mat(abX[100:199]) * RGws02
yHat05 = mat(abX[100:199]) * RGws05

# 分析预测误差的大小，可以用函数rssError计算出这个指标
err01 = rssError(abY[100:199], yHat01.T.A)
err02= rssError(abY[100:199], yHat02.T.A)
err05 = rssError(abY[100:199], yHat05.T.A)
print err01, err02, err05
```

```
423.173802638 429.629980174 460.977796571
```

4.2 前向逐步回归

前向逐步回归算法可以得到与lasso差不多的效果，但更加简单。

一开始，所有的权重都设为1；然后每一步所做的决策是对某个权重增加或减少一个很小的值。

两个输入参数：

- eps：每次迭代需要调整的步长
- numIt：迭代次数

In [37]:
```
# 对特征按照均值为0方差为1进行标准化处理
def regularize(xMat):#regularize by columns
    inMat = xMat.copy()
    inMeans = mean(inMat,0)    #calc mean then subtract it off
    inVar = var(inMat,0)       #calc variance of Xi then divide by it
    inMat = (inMat - inMeans)/inVar
    return inMat
```

In [38]:
```
# 前向逐步线性回归
def stageWise(xArr,yArr,eps=0.01,numIt=100):
    xMat = mat(xArr); yMat=mat(yArr).T
    yMean = mean(yMat,0)
    yMat = yMat - yMean     #can also regularize ys but will get smaller coef
    xMat = regularize(xMat)
    m,n=shape(xMat)
    returnMat = zeros((numIt,n)) #testing code remove
    ws = zeros((n,1)); wsTest = ws.copy(); wsMax = ws.copy()  # 建立两个ws
的副本
    for i in range(numIt):
#     print ws.T
        lowestError = inf; # 误差初始值设为正无穷
        for j in range(n): # 在所有特征上运行两次for循环
            for sign in [-1,1]: # 分别计算增加或减少该特征对误差的影响
                wsTest = ws.copy()
                wsTest[j] += eps*sign
                yTest = xMat*wsTest
```

```
                rssE = rssError(yMat.A,yTest.A)
                if rssE < lowestError:
                    lowestError = rssE
                    wsMax = wsTest
        ws = wsMax.copy()
        returnMat[i,:]=ws.T
    return returnMat
```

In [39]:
```
abX, abY = loadDataSet('data/abalone.txt')
```

In [40]:
```
returnMat = stageWise(abX, abY, 0.01, 200)  # 实验不太步长和步数

#returnMat = stageWise(xArr, yArr, 0.001, 1000) # 实验5000次迭代时间较长
```

In [41]:
```
returnMat
```

Out[41]:
```
array([[ 0.  ,  0.  ,  0.  , ...,  0.  ,  0.  ,  0.  ],
       [ 0.  ,  0.  ,  0.  , ...,  0.  ,  0.  ,  0.  ],
       [ 0.  ,  0.  ,  0.  , ...,  0.  ,  0.  ,  0.  ],
       ...,
       [ 0.05,  0.  ,  0.09, ..., -0.64,  0.  ,  0.36],
       [ 0.04,  0.  ,  0.09, ..., -0.64,  0.  ,  0.36],
       [ 0.05,  0.  ,  0.09, ..., -0.64,  0.  ,  0.36]])
```

观察结果：

1. w1和w6都是0，表明它们不对目标值造成任何影响，也就是说，这些特征很可能是不需要的。
2. 在eps参数设置为0.01的情况下，一段时间后系数就已经饱和并在特定值之间来回震荡，这是因为步长太大。可以看到w0在0.04和0.05之间来回震荡。

下面试验更小的步长和更多的步数：

** 把这些结果与最小二乘法进行比较

In [42]:
```
xMat = mat(abX)
yMat = mat(abY).T
xMat = regularize(xMat)
yMat = yMat - mean(yMat,0)
weights = standRegres(xMat, yMat.T)
weights.T
```

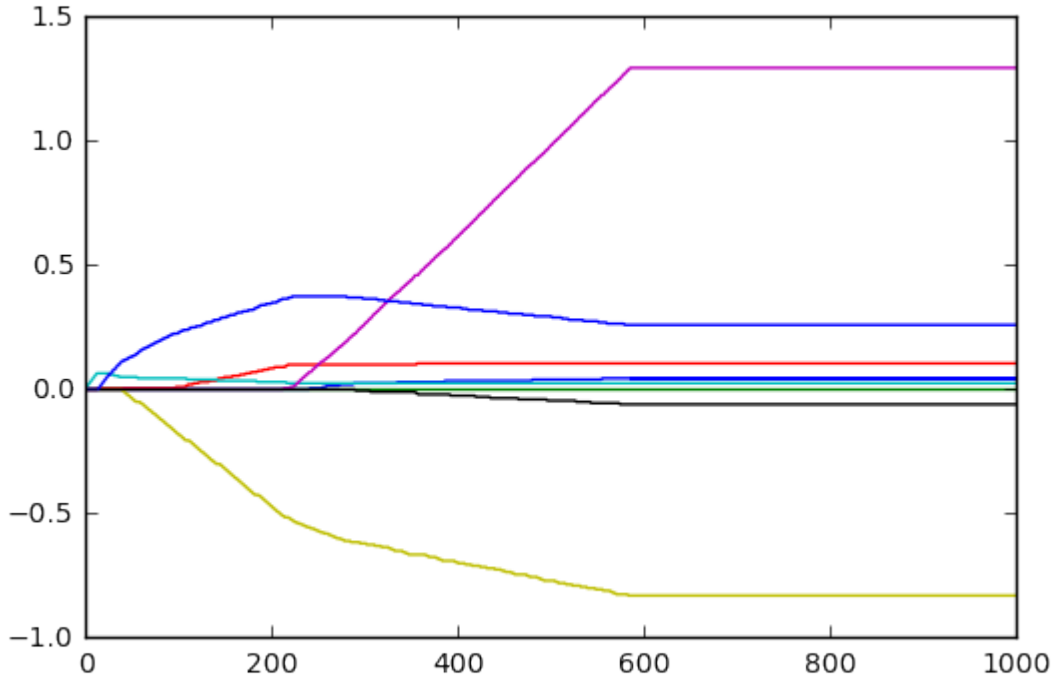
Out[42]:
```
matrix([[ 0.043 , -0.0227,  0.1321,  0.0208,  2.224 , -0.999 , -0.1173,
          0.1662]])
```

** 在5000次迭代之后，逐步线性回归算法与常规的最小二乘法效果类似。

** 下面使用eps=0.005，经过1000次迭代后的结果如图。

In [43]:
```
stageWeights = stageWise(abX, abY, 0.005, 1000) # 实验5000次迭代时间较长
```

In [44]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(stageWeights)
plt.show()
```



** 逐步线性回归得到的系数与迭代次数之间的关系，与lasso的结果相似，但计算起来更加简便。

** 回归算法的好处优点主要在于它可以帮助人们理解现有的模型并做出改进。当构建一个模型后，可以运行该算法找出重要的特征，及时停止对那些不重要特征的收集。

5. 权衡偏差与方差

应用缩减方法（如逐步线性回归和岭回归）时，模型增加了偏差（bias），与此同时却减少了模型的方差。

人工合成一个二维数据，生成公式如下：

y = 3 + 1.7 *x* + *0.1* sin（3x） + 0.06 * N（0，1） N是正态分布

如果使用一条直线拟合，则只能拟合 3 + 1.7 *x* 部分，误差部分就是 *0.1* sin（3x） + 0.06 * N（0，1）

我们测试多组不同的局部权重来找到具有最小测试误差的解。

我们在局部权重线性回归模型中通过引入越来越小的核来不断增大模型的方差。降低核的大小，那么训练误差将变小。

缩减法，可以将一些系数缩减为很小的值或直接缩减为0，这是一个增大模型偏差的例子。通过把一些特征的回归系数缩减为0， 同时也降低了模型的复杂度，使得模型更易理解，同时也降低了预测误差。

方差指的是模型之间的差异，而偏差指的是模型预测值与数据之间的差异。方差是可以度量的，从数据中取随机样本进行线性拟合得到一组回归系数，再取出另外一组随机样本并拟合，得到另外一组回归系数，这些系数之间的差异大小也就是模型方差大小的反映。

6. SKlearn实现Linear Regression

1. 最小二乘法（**Ordinary Least Squares**）的线性回归算法

In [45]:
```
from sklearn import linear_model
clf = linear_model.LinearRegression()
clf.fit ([[0, 0], [1, 1], [2, 2]], [0, 1, 2])
#LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
print clf.coef_
#array([ 0.5,  0.5])
```

```
[ 0.5  0.5]
```

In [46]:
```
# 最小二乘法预测鲍鱼年龄
from sklearn import linear_model
clf = linear_model.LinearRegression()

# 加载数据，分隔为train和test
abX, abY = loadDataSet('data/abalone.txt')
trainX = abX[0:99]; trainY = abY[0:99]
testX = abX[100:199]; testY = abY[100:199]

clf.fit (trainX, trainY) # 训练模型
yHat = clf.predict(testX) # 使用模型去预测
```

In [47]:
```
print yHat[:2]
print testY[:2]
clf.score(testX, testY)  # 返回预测值与真实值相关系数
```

```
[  7.484   10.8339]
[7.0, 15.0]
```

Out[47]:
```
0.53001606224451348
```

In [48]:
```
rssError(array(yHat), array(testY)) # SSE， square sum of error
```

Out[48]:
```
608.50102195578529
```

In [49]:
```
sqrt(rssError(array(yHat), array(testY))) / 100  # 均方误差
```

Out[49]:
```
0.24667813481453627
```

2. 岭回归（**Ordinary Least Squares**）的线性回归算法

In [50]:
```
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression

from sklearn import linear_model
clf = linear_model.Ridge (alpha = .5)
clf.fit ([[0, 0], [0, 0], [1, 1]], [0, .1, 1])
#Ridge(alpha=0.5, copy_X=True, fit_intercept=True, max_iter=None,normalize=False, random_state=None, solver='auto', tol=0.001)
print clf.coef_
#array([ 0.34545455,  0.34545455])
print clf.intercept_
#0.13636...
```

```
[ 0.3455  0.3455]
0.136363636364
```

3. Lasso线性回归算法

In [51]:
```
from sklearn import linear_model
clf = linear_model.Lasso(alpha = 0.1)
clf.fit([[0, 0], [1, 1]], [0, 1])
#Lasso(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=1000,normalize=False, positive=False,
# precompute=False, random_state=None,selection='cyclic', tol=0.0001, warm_start=False)
clf.predict([[1, 1]])
#array([ 0.8])
```

Out[51]:
```
array([ 0.8])
```

4. ElasticNet线性回归算法

class sklearn.linear_model.ElasticNet(alpha=1.0, l1_ratio=0.5, fit_intercept=True, normalize=False, precompute=False, max_iter=1000, copy_X=True, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic')

a *L1* + *b* L2

alpha = a + b

l1_ratio = a / (a + b)

In [52]:
```
from sklearn import linear_model
clf = linear_model.ElasticNet(alpha = 1.0)
clf.fit([[0, 0], [1, 1]], [0, 1])
#Lasso(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=1000,normalize=False, positive=False,
# precompute=False, random_state=None,selection='cyclic', tol=0.0001, warm_start=False)
clf.predict([[1, 1]])
#array([ 0.8])
```

Out[52]:
```
array([ 0.5])
```

5. SGDRegressor回归算法

In [53]:
```
import numpy as np
from sklearn import linear_model
n_samples, n_features = 10, 5
np.random.seed(0)
y = np.random.randn(n_samples)
X = np.random.randn(n_samples, n_features)
clf = linear_model.SGDRegressor()
clf.fit(X, y)
#SGDRegressor(alpha=0.0001, average=False, epsilon=0.1, eta0=0.01,
#        fit_intercept=True, l1_ratio=0.15, learning_rate='invscaling',
#        loss='squared_loss', n_iter=5, penalty='l2', power_t=0.25,
#        random_state=None, shuffle=True, verbose=0, warm_start=False)
clf.predict([[1, 1, 1, 1, 1]])
```

Out[53]:
```
array([ 0.1536])
```

实例**2**：预测台风路径

台风数据data/typhoon.dat, 共计500条，共18列，前16列为预报因子，最后2列分别是经纬度。

每行数据是间隔12个小时采集到的台风预报因子，数据的任务是预测台风的路径，即预测台风的经

纬度，最后两列的值。

- LON .V1： 起报时刻经度 lon+00
- LON .V2： 起报时刻前12小时纬向移速 vlat-12
- LON .V3： 起报时刻前24小时纬向移速 vlat-24（可能是经向）
- LON .V4： 起报时刻前24 小时所在经度 lon-24
- LON .V5： 起报时刻前12 小时至前24小时纬向移速 vlat-24
- LON .V6： 起报时刻与前12小时的经度差 (lon+00)-(lon-12)
- LON .V7： 起报时刻前6 小时所在经度 lon-06
- LON .V8： 起报时刻前18 小时所在经度 lon-18
- Lat.v9: 起报时刻纬度 lat+00
- Lat.v10: 起报时刻前12小时经向移速 vlon-12
- Lat.v11: 起报时刻前24小时经向移速的平方 (vlon-24)^2
- Lat.v12: 起报时刻前12 小时所在纬度 lat-12
- Lat.v13: 起报时刻与前24小时的经度差 (lon+00)-(lon-24)
- Lat.v14: 起报时刻前6 小时所在纬度 lat-06
- Lat.v15: 起报时刻前18 小时所在纬度 lat-18
- Lat.v16: 起报时刻前6 小时地面附近最大风速 wind-06
- LON . t： 要预报的24小时后的经度（预报量） lon+24
- Lat.t: 要预报的24小时后的纬度（预报量） lat+24

In [54]:
```
from __future__ import division
import os, sys
from numpy import *
from math import *

#加载数据
def loadDataSet(fileName):       #general function to parse tab -delimited floats
    numFeat = len(open(fileName).readline().split('\t')) - 2 #get number of fi
elds
    dataArr = []
    lonArr = []
    latArr = []
    fr = open(fileName)
    for line in fr.readlines():
        lineArr =[]
        lineArr.append(1.0)  # 增加这一列是为了进行线性回归，存放常量x0
        curLine = line.strip().split('\t')
        for i in range(numFeat):
            lineArr.append(float(curLine[i]))
        dataArr.append(lineArr)
        lonArr.append(float(curLine[-2]))  # 倒数第二项是lon经度
        latArr.append(float(curLine[-1]))  # 最后一项是lat纬度
    return dataArr,lonArr, latArr
```

1） 标准回归训练

In [55]:
```
# 1. 标准回归训练
def standRegres(xArr,yArr):
    xMat = mat(xArr); yMat = mat(yArr).T
    xTx = xMat.T*xMat
    if linalg.det(xTx) == 0.0:
        print "This matrix is singular, cannot do inverse"
        return
    ws = xTx.I * (xMat.T*yMat)
```

```
    return ws
```

In [56]:
```
dataArr,lonArr,latArr= loadDataSet('data/typhoon.dat')
```

In [57]:
```
#计算误差
def absError(yArr,yHatArr): #yArr and yHatArr both need to be arrays
    return abs(yArr-yHatArr).sum() / len(yArr)
```

In [58]:
```
#标准回归预测台风经度回归模型的参数
wsLon = standRegres(dataArr,lonArr)
wsLon
```

Out[58]:
```
matrix([[ -1.5467e+00],
        [  3.7212e+01],
        [ -2.1843e+00],
        [  1.4225e+00],
        [ -3.2901e+01],
        [ -1.9341e-01],
        [  1.6794e+01],
        [ -2.5340e+00],
        [ -5.7848e-01],
        [  2.4633e+03],
        [ -2.6876e+02],
        [  2.0940e-03],
        [ -2.4631e+03],
        [ -3.7468e+01],
        [  1.1001e-01],
        [ -1.9900e-01],
        [ -1.9233e-02]])
```

In [59]:
```
#标准回归预测台风纬度
wsLat = standRegres(dataArr,latArr)
wsLat
```

Out[59]:
```
matrix([[  3.9490e+00],
        [  3.5525e+01],
        [  6.5507e+00],
        [ -1.2969e+01],
        [ -3.4416e+01],
        [  6.4507e+00],
        [ -1.2002e+00],
        [ -9.1449e-01],
        [ -1.8375e-01],
        [  2.5566e+03],
        [ -2.7858e+02],
        [ -3.9001e-04],
        [ -2.5543e+03],
        [ -3.4225e+01],
        [ -1.5890e+00],
        [  1.0309e-01],
        [  1.6060e-02]])
```

In [60]:
```
#根据回归模型计算预测台风经度
xMat = mat(dataArr)
```

In [61]:
```
#预测经度值
lonHatArr = xMat * wsLon
```

In [62]:
```
#预测纬度值
latHatArr = xMat * wsLat
```

In [63]:
```
#经度误差
absErrorLon = absError(lonArr,lonHatArr.T)
absErrorLon
```

Out[63]:
```
22.872156307980347
```

In [64]:
```
#纬度误差
absErrorLat = absError(latArr,latHatArr.T)
absErrorLat
```

Out[64]:
```
2.259120802758054
```

In [65]:
```
#计算经度预测值与真实值的相关系数
corrcoef(lonHatArr.T, lonArr)
```

Out[65]:
```
array([[ 1.    ,  0.9525],
       [ 0.9525,  1.    ]])
```

In [66]:
```
#计算纬度预测值与真实值的相关系数
corrcoef(latHatArr.T, latArr)
```

Out[66]:
```
array([[ 1.    ,  0.9208],
       [ 0.9208,  1.    ]])
```

In [67]:
```
#计算距离公式：  110*sqrt(lon^2+lat^2)
110 * sqrt(absErrorLon**2 + absErrorLat**2)
```

Out[67]:
```
2528.179947676883
```

2） 调用**Sklearn**的回归模型

In [68]:
```
import numpy as np
from sklearn import linear_model

#加载数据，分隔为train和test
dataArr,lonArr,latArr= loadDataSet('data/typhoon.dat')
trainX=dataArr[0:299]; lonTrainY=lonArr[0:299]; latTrainY=latArr[0:299]
testX=dataArr[300:]; lonTestY=lonArr[300:]; latTestY=latArr[300:]

#构建模型
from sklearn import linear_model
clf = linear_model.LinearRegression()  #线性回归
#clf = linear_model.Ridge (alpha = 0.8) #岭回归
#clf = linear_model.ElasticNet(alpha = 1.0) #ElasticNet线性

clf.fit (trainX, lonTrainY) #训练经度模型
lonyHat = clf.predict(testX) #使用经度模型去预测经度

clf.fit (trainX, latTrainY) #训练纬度模型
latyHat = clf.predict(testX) #使用纬度模型去预测纬度
```

In [69]:
```
#经度误差
absErrorLon = absError(lonTestY,lonyHat)
#纬度误差
absErrorLat = absError(latTestY,latyHat)
```

```
#计算距离公式：  110*sqrt(lon^2+lat^2)
110 * sqrt(absErrorLon**2 + absErrorLat**2)
```

Out[69]: 158.5081980464272

作业**3**

- 台风路径预测：运用并比较多种线性回归策略提高台风路径预测的准确性